photPARTY: Python Automated Square-Aperture Photometry

By

Teresa A. Symons

B.S., Embry-Riddle Aeronautical University, 2014

Submitted to the graduate degree program in the Department of Physics and Astronomy and the Graduate Faculty of the University of Kansas in partial fulfillment of the requirements for the degree of Master of Science with emphasis in Computational Physics and Astronomy.

_____

Chair: Dr. Barbara J. Anthony-Twarog

_____

Dr. Hume A. Feldman

_____

Dr. Graham W. Wilson

Date Defended: April 17, 2017

The thesis committee for Teresa A. Symons certifies that this is the approved version of the following thesis:

photPARTY: Python Automated Square-Aperture Photometry

Chair: Dr. Barbara J. Anthony-Twarog

Date Approved: April 24, 2017

Abstract

As CCD's have drastically increased the amount of information recorded per frame, so too have they increased the time and effort needed to sift through the data. For observations of a single star, information from millions of pixels needs to be distilled into one number: the magnitude. Various computer systems have been used to streamline this process over the years. The CCDPhot photometer, in use at the Kitt Peak 0.9-m telescope in the 1990's, allowed for user settings and provided real time magnitudes during observation of single stars. It is this level of speed and convenience that inspired the development of the Python-based software analysis system photPARTY, which can quickly and efficiently produce magnitudes for a set of single-star or un-crowded field CCD frames. Seeking to remove the need for manual interaction after initial settings for a group of images, photPARTY automatically locates stars, subtracts the background, and performs square-aperture photometry. Rather than being a package of available functions, it is essentially a self-contained, one-click analysis system, with the capability to process several hundred frames in just a couple of minutes. Results of comparisons with present systems such as IRAF are presented.

Acknowledgements

This work would not have been possible without funding from the National Science Foundation through grant AST 1211621, as well as from the University of Kansas. Many thanks to Kaitlin Neill, an undergraduate student who performed IRAF comparison photometry, as well as Dr. Hume Feldman and Dr. Graham Wilson, who gave their time to be members of my thesis committee. I am especially grateful to Dr. Barbara Anthony-Twarog for her constant guidance, support, and assistance, all of which have been essential in helping me to complete my degree. Thanks to my parents, whose love and encouragement have allowed me to succeed throughout my education, and lastly to Ross, who is my ceaseless source of strength and serenity.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Introduction

With the introduction of CCDs to astronomical imaging decades ago, the capability of every telescope has been enhanced. However, there are still contexts when millions of pixels and thousands of megabytes must be processed to produce measurements for one or a few stars, often with the end goal of reaching a single number: a star's magnitude, which is a logarithmic measure of its brightness. Many software systems and methods have been developed to simplify and accelerate the process of astronomical data analysis. The goal of this work was to examine several currently available options and ultimately to develop an automated program for the

detection of stars and determination of magnitudes for those stars. The underlying inspiration of this project was to recreate the efficiency of the Kitt Peak National Observatory's CCDPhot system, in use on the 0.9-m telescope in the 1990s. The system used a small region-of-interest on a CCD to generate real-time magnitude measurements for single stars (Tody & Davis, 1992). These goals inspired the development of photPARTY, a Python 3.5 program consisting of one main script and six auxiliary functions. The package was designed to be as hands-free as possible and can produce magnitudes for a set of frames autonomously after initial selection of user parameters. For each image file within a designated directory, an output file is created with a simple table of all located stars' positions, magnitudes, and magnitude errors.

1.2 Photometry

Aperture photometry is the process of defining an aperture of desired shape and size around some astronomical object, summing the flux within that aperture, subtracting the flux contribution of the background sky for the same region by estimating the flux of a neighboring region of blank sky, and then converting the star's flux into an instrumental magnitude (Wells, 1994). Apertures are typically circular, which requires them to be precisely placed such that the star is in the center. The radiant flux of an object is a measure of the object's brightness as a result of the total light energy received by some detector. Flux can be defined as:

$$F = \frac{L}{4\pi r^2}$$

where L is an object's inherent luminosity and r is the distance at which the flux, F, is measured. The magnitude system provides a method of determining an object's brightness relative to other objects. Originally based on the idea that the human eye responds logarithmically to differences

in brightness, the scale is set such that a difference of five magnitudes corresponds to a factor of 100 in brightness, with smaller magnitudes indicating brighter objects. This provides the following ratio between two objects' fluxes:

$$\frac{F_2}{F_1} = 100^{(m_1 - m_2)/5}$$

where m_1 and m_2 are the objects' apparent magnitudes.

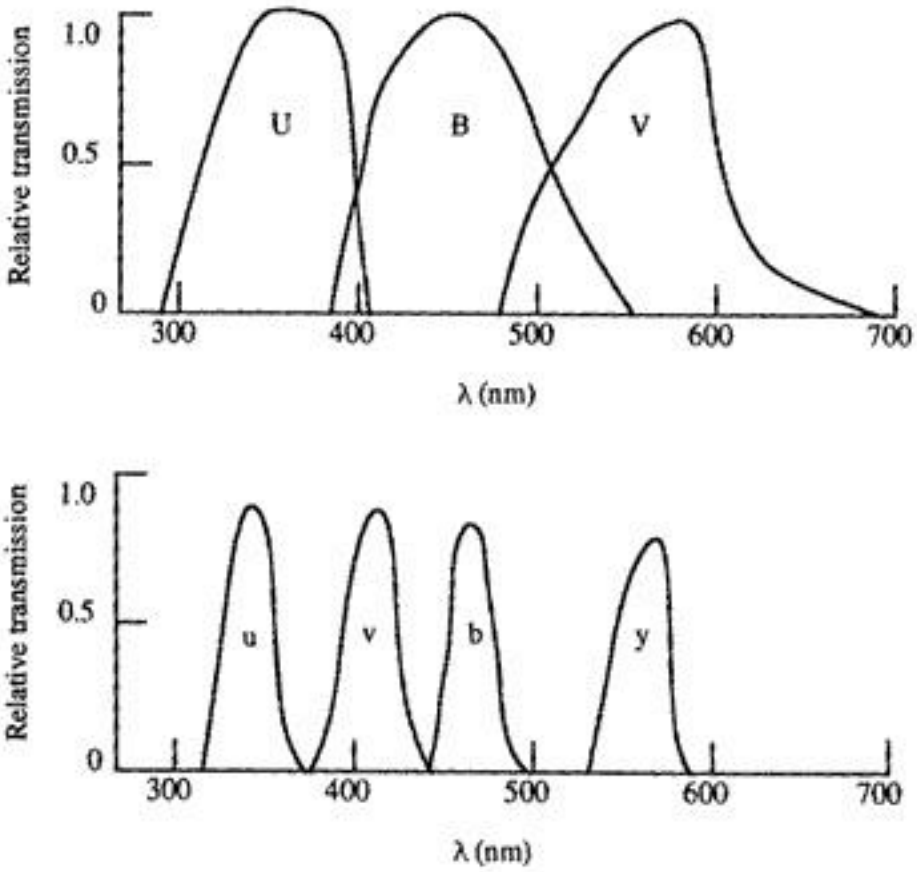

Figure 2: Comparison between Johnson (top) and Stromgren (bottom) bandpass filter systems.

This equation can be stated another way:

$$m_1 - m_2 = -2.5 log_{10}(\frac{F_1}{F_2})$$

These magnitudes correspond to the specific bandpass or wavelength of light that was detected. For each exposure, a single filter is often used to select a specific wavelength band. There are many different types of filter systems. The UBV system, or Johnson system, contains three wide filter bands, each with a central wavelength and effective bandwidth. The U or ultraviolet filter is centered at 365 nm with a width of 68 nm, the B or blue filter is centered at 440 nm with a width of 98 nm, and the V or visual filter is centered at 550 nm with a width of 89 nm (Carroll & Ostlie, 2007). Another commonly used system is the Stromgren intermediate band system, which contains narrow filters u, v, b, and y. The u filter is centered at 350 nm with half-width 30 nm, the v filter is centered at 411 nm with half-width 19 nm, the b filter is centered at 467 nm with half-width 18 nm, and the y filter is centered at 547 nm with half-width 23 nm (Stromgren, 1966). A comparison of the two filter systems can be seen in Figure 1. Measuring a star with more than one filter allows for a better determination of the star's intrinsic color (Carroll & Ostlie, 2007).

Modern CCD frames provide an easy measure of an object's flux as the entire frame is a grid of pixels, each with a digital number proportional to the number of photons collected. Flux can be found by dividing the photon count by the frame's exposure time. There are several software analysis systems presently available that provide a framework for locating stars within CCD frames, selecting apertures, and calculating magnitudes. The oldest and most widely used of these is IRAF, the Image Reduction and Analysis Facility (Tody, 1986).

1.3 IRAF

IRAF was developed at Kitt Peak National Observatory beginning in 1981. It serves as a software system for scientific data reduction and analysis, with a large number of programs devoted to optical astronomy data analysis (Tody, 1986). Several packages and options exist for star detection and aperture photometry. In particular, APPHOT is a package of tools for uncrowded field photometry and DAOPHOT is better suited toward crowded fields with many objects. Both packages offer the *daofind* program to locate stars, which searches for areas with a specified full-width half-maximum and a peak amplitude above a defined background level. The *phot* program takes a list of located stars and computes their centers, determines a background sky level for each, and calculates their magnitudes using defined apertures (Wells, 1994). There are extensive user-defined parameters that affect how each of these steps is performed. For a set of single stars, final magnitudes are relatively insensitive to user-selected parameters. An alternative to aperture photometry is point spread function (PSF) fitting. The *psf* program computes a PSF model from a list of stars. Programs like *peak*, *nstar*, or *allstar* fit the PSF model to individual stars or groups of stars, resulting in relative magnitudes for all the stars in even a very crowded field. A standard or photometric magnitude is still required to transform the relative magnitudes to true values (Davis, 1994).

1.4 Astropy

In recent years, the Python programming language has become very widely adopted not only in astronomy, but also many other scientific fields. The community-developed Astropy Python

package, initially released in 2013, represents the largest comprehensive astronomical analysis suite for Python. Astropy contains much of the basic framework for working with astronomical data formats and coordinate systems, and for performing basic data visualization and analysis (The Astropy Collaboration, 2013). In addition, many affiliate packages are available for various purposes separate from those that are provided with the core Astropy package. One in particular, Photutils, provides the tools for different types of photometry. Among others, it contains programs for background level calculation, source detection, and aperture photometry. Several of these are Python implementations of IRAF algorithms (Bradley et al., 2016).

Chapter 2

Program Development

2.1 Program Overview

PhotPARTY was developed in Python 3.5 using the PyCharm Community Edition IDE. It makes use of common science and analysis packages such as NumPy, SciPy, and Matplotlib, as well as Astropy, all installed through the Anaconda Python distribution. The main script, *photparty.py*, contains user settings and calls to six auxiliary functions. This script may be found in Appendix A. Once settings are chosen, simply running the script will locate stars and return magnitudes for a set of CCD frames. The program is primarily designed to handle single-star or uncrowded field

frames, and frames should already be pre-processed with instrumental signatures removed and corrections made.

Each CCD frame contains a two-dimensional array of values where each cell contains a single pixel's photon count. Additionally, information is stored in a frame header that provides details about the CCD chip and the type of exposure taken for that particular frame. This information can be retrieved using specified keywords for each value. All .fit or .fits files in the user-defined directory of choice will be analyzed and output files are written to the same directory. These are the typical file types for CCD frames.

The frames are processed in a loop that first retrieves a given file's data and header information such as exposure time and filter information. Either the entire frame is used or a smaller portion is taken if the user desires, and pixel values above and below a user-defined threshold are zeroed out to prevent defects such as cosmic rays being detected as stars. The sky background level is calculated for the data inset area and used to determine where stars exist. Optionally, plots may be displayed at several points throughout processing for a more interactive experience. After stars are located with paired X and Y coordinates, photometry is performed. The results are then written to text files.

2.2 User Parameters

Extensive user parameters are defined at the beginning of the main script. These parameters must ideally be altered before running the program on a set of frames sharing an individual CCD on a given night, but may need to be customized for sets of frames of the same object or objects. The user must specify the path of the directory containing the frames to be analyzed, as well as the

header keywords for exposure time, filter type, airmass, and gain, which is the factor that scales between detected photons and recorded digital units or counts. If no keyword is available in the header, the user may choose to specify ‘none’ and input a value manually. Optional parameters include the number and size of boxes to be used for random sampling in determination of the background sky level. The user has three options in selecting the desired data inset of the frame: ‘whole,’ ‘half,’ and ‘custom.’ The program can either inspect the entire frame, the inner 50% of the frame, or a custom specified area in which the user inputs beginning and ending X and Y coordinates based on (1,1) as the lower left-hand corner of the frame as viewed in SAOImage DS9. This is a common program used for viewing astronomical images.

The user may specify custom pixel-rejection parameters, including an upper bound above which all pixel values are set to 0, and a number of standard deviations below which negative pixel values will be set to 0. Negative pixel values may occur after pre-processing, during which values may be subtracted from the overall frame. Pixels with values reset to 0 are still counted and used during various calculations, introducing the possibility of skewed values. Probably the most important parameter is the sigma detection level or the number of standard deviations above the calculated sky background necessary for star detection. The user may choose to suppress or display plots of the chosen detection level juxtaposed with the frame data to make clear what will be detected as a star and what will not. Likewise, the user may also suppress or display a plot of the frame image with square apertures around the stars being analyzed. If the user chose to select a custom area of interest, that area will be boxed in this plot as well. If plots are not suppressed, they will display for every frame being analyzed and program function will pause until they are dismissed. Lastly, the user may select the size of the square aperture to be used for photometry.

2.3 Modules

The six auxiliary functions or modules of the program are referenced by the main script as needed and do not need to be accessed or modified in any way for the program to function. Their purpose and methods are described in detail below.

2.3.1 Index Adjustment

The function *fixindex.py* is not a main program module, but is used by two other modules: *background.py* and *starphot.py*. At several times, random sub-arrays are selected from the main data inset. The lower left-hand corner is randomly selected within the data inset array, but the possibility exists for the area of the sub-array to extend outside the boundaries of the data inset. Similarly, if a star is located near one of the inset boundaries, it is possible for photometric apertures to extend outside the defined boundaries. This function checks if the edges of any smaller box fall within the boundaries of some larger box. If they do not, they are adjusted to match the nearest edge. The adjusted indices are returned and accepted as the new boundaries of the sub-array. This function can be found in Appendix B.

2.3.2 Background Calculation

As a first step in analysis, the *background.py* function computes the background sky level for each particular frame through median sampling. This function takes the selected data inset and randomly selects a number of smaller sub-arrays, calculating the median pixel value for each

one. The number and size of sub-arrays is defined by the user. All sampling arrays are checked and adjusted by the *fixindex.py* function. After sampling is finished, the median of all sub-array medians is taken as the background level for the frame in question. This background level is used as a basis to determine the location of stars. This function can be found in Appendix C.

2.3.3 Pixel Summation

In the next step, the selected data inset of each frame is summed by row (y-coordinate) and by column (x-coordinate) by the *binsum.py* function. All pixel values in each row and each column

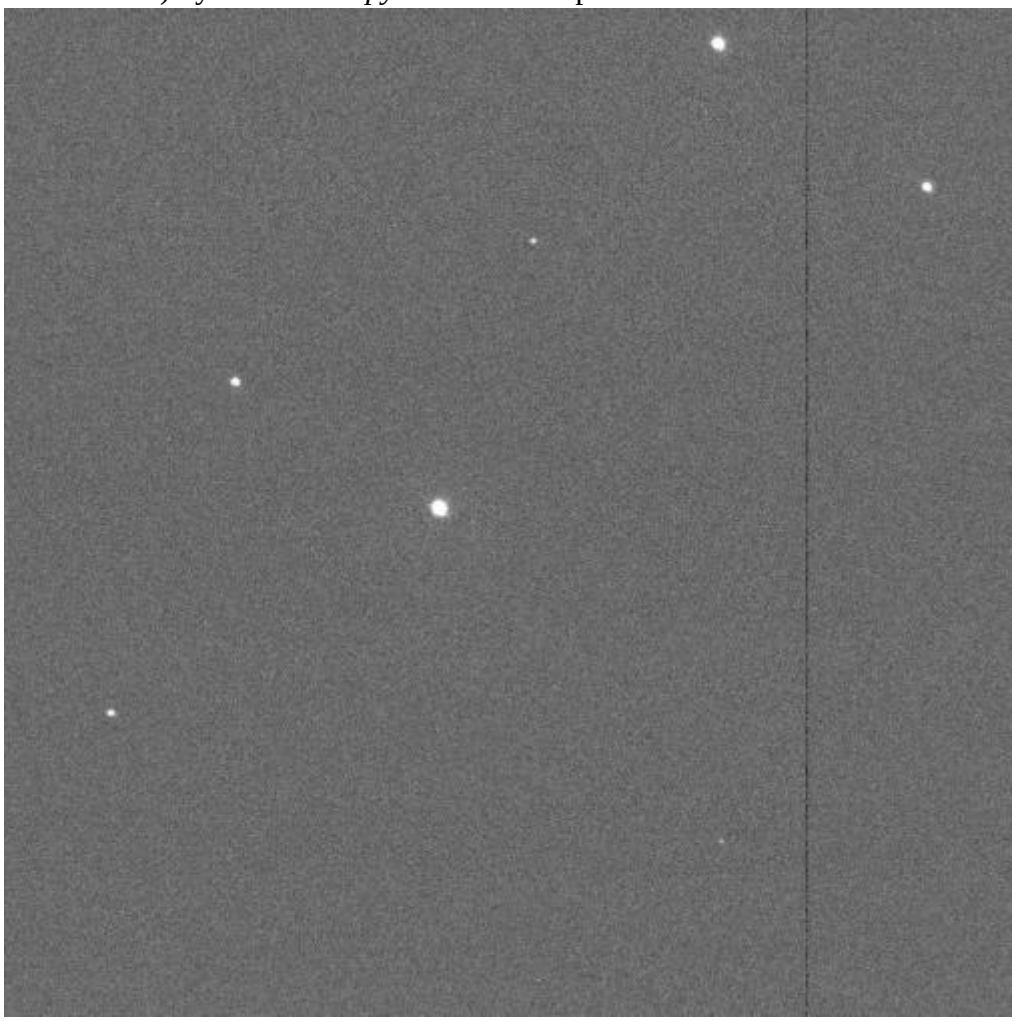

Figure 3: An example CCD frame processed by photPARTY. This frame was taken in October 2016 with the 40-inch telescope at the Mount Laguna Observatory. The total frame has a length and width of 2048 pixels.

are summed and saved as vectors. These vectors are used to initially locate stars, which will appear as summed rows and columns with peak values above the background level. This function can be found in Appendix D.

2.3.4 Star Detection

The *starlocate.py* function takes the summed row and column vectors and determines which rows and columns contain potential stars. First, the sky background level of the user-selected data inset is multiplied by the length of one of the dimensions of the inset in order to determine the approximate pixel count of a summed row or column of pure sky. Any row or column with a star must have a value above this base level. The user may choose the number of standard deviations above the background level required for star detection. The standard deviation of all pixels in the data inset is calculated, and the desired number of standard deviations above the summed background is taken as the detection level. All rows and columns above the detection level are flagged as potentially having stars. If no stars are found in either dimension, an error is returned suggesting that the user examine the detection level. Optionally, a plot is displayed showing the summed row and column values with the detection level overlaid, providing an easy way to determine which peaks will and will not be flagged as stars under the current detection level.

Figure 1 shows an example of a CCD frame that was processed by photPARTY. On the right-hand side of the frame, a defect is clearly seen. Given that the bright star near the center of the frame is the likely target of the image, the ‘half’ option was chosen for the data inset, using only the inner 50% of the frame to avoid the defect on the right. After the starlocate.py function

had processed the frame, Figures 2 and 3 were displayed. They show the summed row and column values for the frame, where the detection level is shown in green. Any peaks above the detection level will be flagged as potential stars, while any underneath will be ignored. The largest peak represents the bright, nearly central star. These plots provide the user a visual means of determining if an appropriate detection level has been chosen, and can be suppressed once quick analysis of many frames is desired. This function can be found in Appendix E.

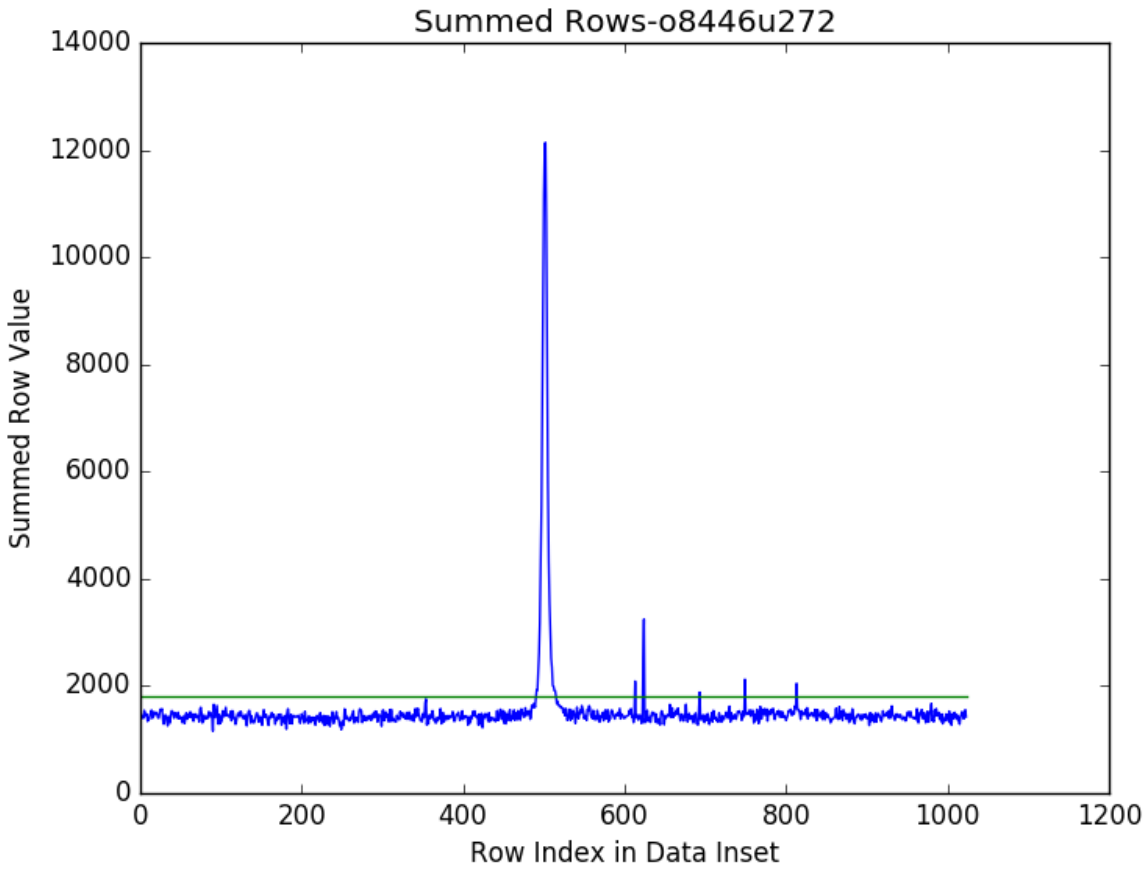

Figure 4: Summed row values versus row numbers for the example CCD frame. The green line indicates the detection level, above which stars will be detected.

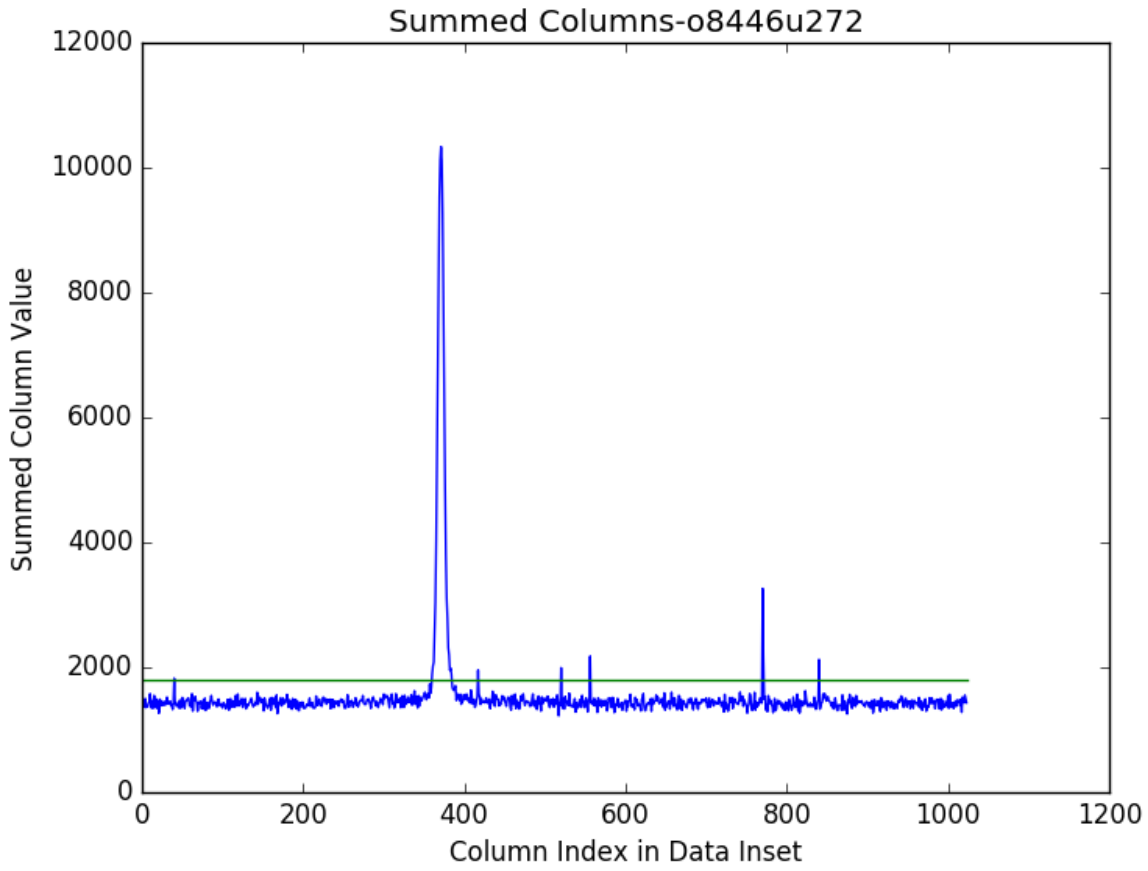

Figure 5: Summed column values for the same example frame.

2.3.5 Star Coordinates

The *starmed.py* function takes the lists of rows and columns that may contain stars and turns them into paired sets of coordinates for specific stars. Each list contains the numbers or indices of all rows or columns that may contain stars, but stars are generally more than a single row or column in width. Because stars are typically between five and 15 pixels wide, each list contains

different sets of indices all belonging to the same individual stars. Unless separated by star, each individual pixel of a star would be evaluated as a separate star. This function takes each list and searches for a minimum of one pixel gaps between indices. If stars do not have a one pixel gap in either dimension, they cannot be separated and evaluated as two separate stars. Frames with stars that are very close together in either dimension cannot be accurately evaluated. Each time a gap is found, a new sub-list is created belonging to a new star. The single row and column lists are divided into separate lists for each star. Then, a median row and column pixel is calculated for each star. If a star has only one pixel in either dimension, that pixel is taken as the central coordinate. This creates a list of single row and column coordinates where the centers of stars may be found.

The final step involves pairing row and column coordinates together such that they match. For each row where a star's center has been found, a search is performed in the original frame data for the maximum column value associated with that row. It is assumed that in a row where a star is known to exist, the maximum column value will match the location of that star. This maximum column value is compared to the list of all known columns with stars. If the value is present within an error of five pixels to allow for an offset in the star's center, the row and column coordinates are taken as a matched pair. This does not allow for matches to be made in the case of a single row or column containing two exact star centers. However, this case is very rare in an uncrowded frame. A final list of paired star coordinates is prepared and adjusted to reflect the entire frame if a subset of the frame was chosen. This function can be found in Appendix F.

2.3.6 Photometry

For each star with matched coordinates, the final analysis step is performing photometry to determine the magnitude of the star. The function *starphot.py* places a square box of user-defined size around each star, referencing the *fixindex.py* function to make adjustments if the box boundaries exceed the edges of the data inset. Four additional boxes of the same size are placed at the four corners of the star's box, with similar adjustments being made if necessary. The median pixel values of the four corner boxes are calculated and the median of those medians is taken as a local sky background value for each particular star. The pixels inside the star's square aperture are summed and the background value is subtracted. At this point any negative summed values will not result in a magnitude, and if any are present, an error is printed advising the user which frame or frames contain these values and need more careful inspection. The background-subtracted pixel counts are divided by the frame's exposure time to convert the counts into fluxes. These fluxes are then converted to magnitudes using

$$m = -2.5 * log_{10}(F) + 20$$

where F is the flux, or the number of photons per second, and 20 is an arbitrary offset value chosen to make the magnitudes positive. An error in each magnitude is also calculated using

$$m_{err} = \frac{1}{\sqrt{n_\gamma}}$$

where n_γ is the background-subtracted, summed, detected photon count for the star. The basis for this error is the assumption that counting photons can be described by Poisson statistics. The magnitude error shown above is the fractional uncertainty in counting the number of photons

(Chromey, 2016). If the user desires, a plot will display for each frame showing each detected star for which photometry was performed. An example of this can be seen in Figure 4.

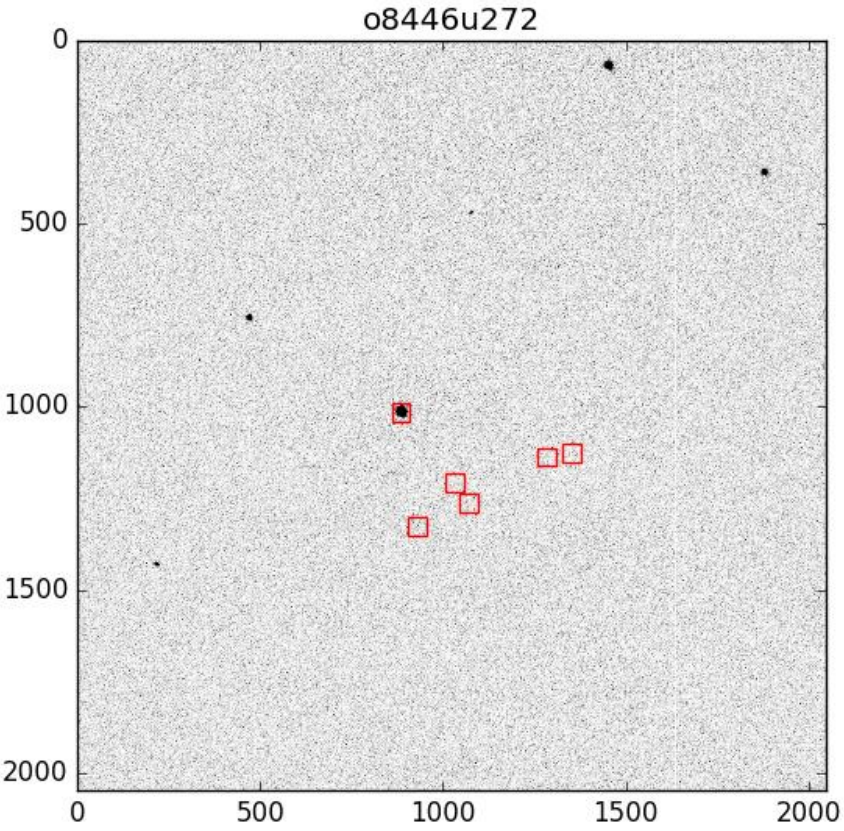

Figure 6: The same example frame with square apertures overlaid on detected stars.

The red boxes show the specific location and size of the square apertures used to perform photometry and determine magnitudes for the detected stars. Note that only stars in the central portion of the frame have been detected as per the parameters chosen. Closer inspection in Figure 5 reveals that each box contains a star. This function can be found in Appendix G.

2.4 Sample Output

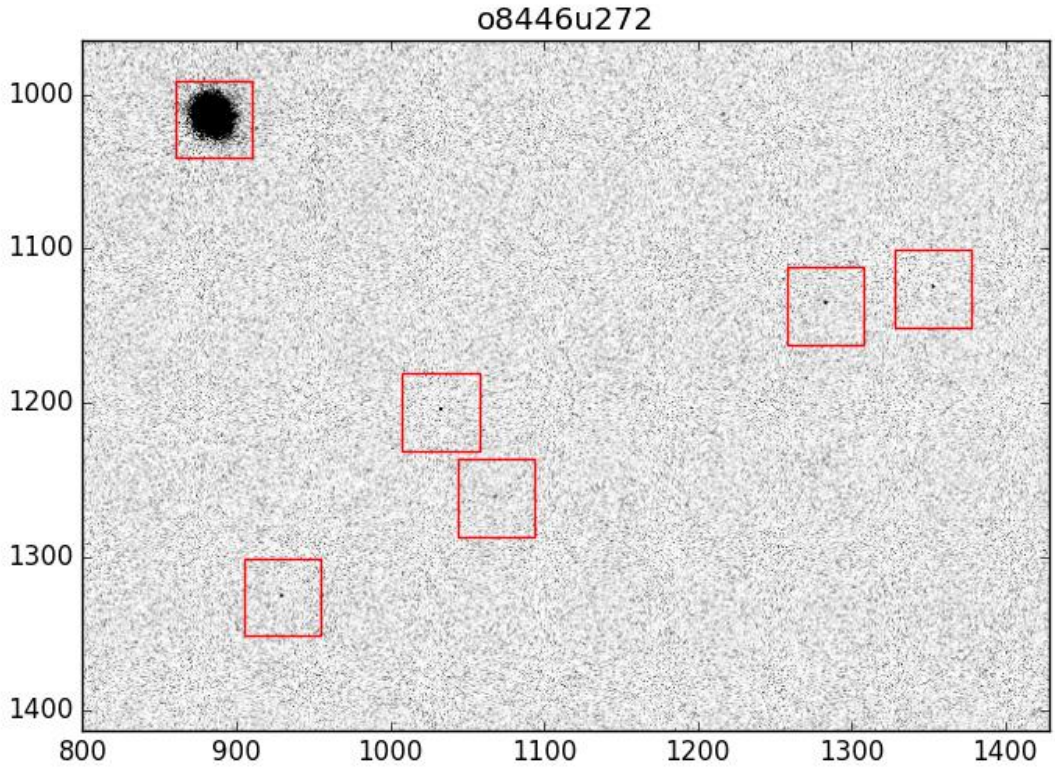

Figure 7: A closer look at the detected stars from the previous figure. Flux for all pixels within the red square apertures will be added to determine magnitudes for those stars.

Once *starphot.py* has completed calculation of the magnitudes and the plot has been examined if desired, two text files are written to the same directory as the CCD frames. Both will contain the name of the original frame to which they pertain. The first ends with 'dat.txt' and is the primary output of the program. It contains a table listing all of the stars found with the magnitudes and magnitude errors. This can be seen in Figure 6. For each star, the original file name and header information including filter type, airmass, and exposure time are listed. Coordinates for the star are given in the context of the original frame following (1,1) as the lower left-hand corner. In the

event of a negative background-subtracted pixel count for a particular star, a ‘nan’ value will appear for the magnitude and the magnitude error. The second output text file ends with ‘info.txt’ and is meant primarily for debugging and troubleshooting purposes. It lists directly the output of many intermediary analysis steps and can give the user a better sense of how the steps were carried out and where any problems may exist. The info output file for the example frame can be seen in Figure 7.

```
File_Name Filter Airmass Exposure_Time X Y Magnitude Mag_Err
o8446u272.fit u 1.465521 25.0 885 1016 11.12029 0.00335
o8446u272.fit u 1.465521 25.0 1353 1126 16.48160 0.03956
o8446u272.fit u 1.465521 25.0 1283 1137 14.83805 0.01856
o8446u272.fit u 1.465521 25.0 1033 1206 16.73343 0.04443
o8446u272.fit u 1.465521 25.0 1069 1262 15.92787 0.03066
o8446u272.fit u 1.465521 25.0 930 1326 16.23189 0.03527
```

Figure 8: Sample output text file for the same example frame.

```
Exposure time:
25.0
Filter:
u
Airmass:
1.465521
Sky background level:
1.40089
Shape of inset array:
(1024, 1024)
Inset background sky level:
1.3766
Summed background value for one row/column:
1409.63720703
Standard deviation of inset:
7.28751
Detection level in sigma:
50
Indices of detected stars:
[491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507,
508, 509, 510, 511, 512, 513, 514, 515, 613, 623, 624, 693, 749, 813]
[40, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375,
376, 377, 378, 379, 380, 381, 382, 383, 384, 417, 520, 556, 770, 771, 840]
Indices of detected stars divided into sublist by star:
[[491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507,
508, 509, 510, 511, 512, 513, 514, 515], [613], [623, 624], [693], [749], [813]]
[[40], [360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375,
376, 377, 378, 379, 380, 381, 382, 383, 384], [417], [520], [556], [770, 771], [840]]
Number of stars found by row/column:
6
7
Median pixel of each star by row/column:
[503, 613, 624, 693, 749, 813]
[40, 372, 417, 520, 556, 770, 840]
Paired indices of star centers:
[[503, 372], [613, 840], [624, 770], [693, 520], [749, 556], [813, 417]]
Total number of stars found:
6
Coordinates of stars (x,y):
[[885, 1016], [1353, 1126], [1283, 1137], [1033, 1206], [1069, 1262], [930, 1326]]
Width/Height of boxes:
50
Pixel sums for boxes around each star:
[93075.859, 4029.252, 6250.2373, 4096.127, 4585.6152, 4360.1943]
Background values for each star:
[3987.4941110610962, 3390.5479311943054, 3348.09809923172, 3589.6444320678711,
3521.982729434967, 3556.3349723815918]
Background subtracted star values:
[89088.365263938904, 638.70402193069458, 2902.13920545578, 506.48252105712891,
1063.632504940033, 803.8593635559082]
Flux of stars:
[3563.5346105575563, 25.548160877227783, 116.0855682182312, 20.259300842285157,
42.545300197601321, 32.154374542236326]
```

Figure 9: Secondary output text file, containing step-by-step results for many intermediary segments of the program.

Chapter 3

Discussion of Results

3.1 Comparison with IRAF

In order to gauge the accuracy of photPARTY's methods, a set of 335 frames were analyzed by both photPARTY and manually in IRAF. A direct comparison of the magnitude values can be seen in Figure 8. The linear relationship indicates that the values are in very good agreement, with an arbitrary offset of five due to photPARTY adding an arbitrary offset of 20 to all magnitudes, while IRAF adds 25. Figure 9 shows the difference between the magnitudes from IRAF aperture photometry and those from photPARTY over the range of IRAF values. Due to the arbitrary offset, five is essentially the zero point for the comparison. Most magnitudes agree

within ±0.02. The mean difference is 5.00 with 0.01 root mean square, indicating a consistency in the two

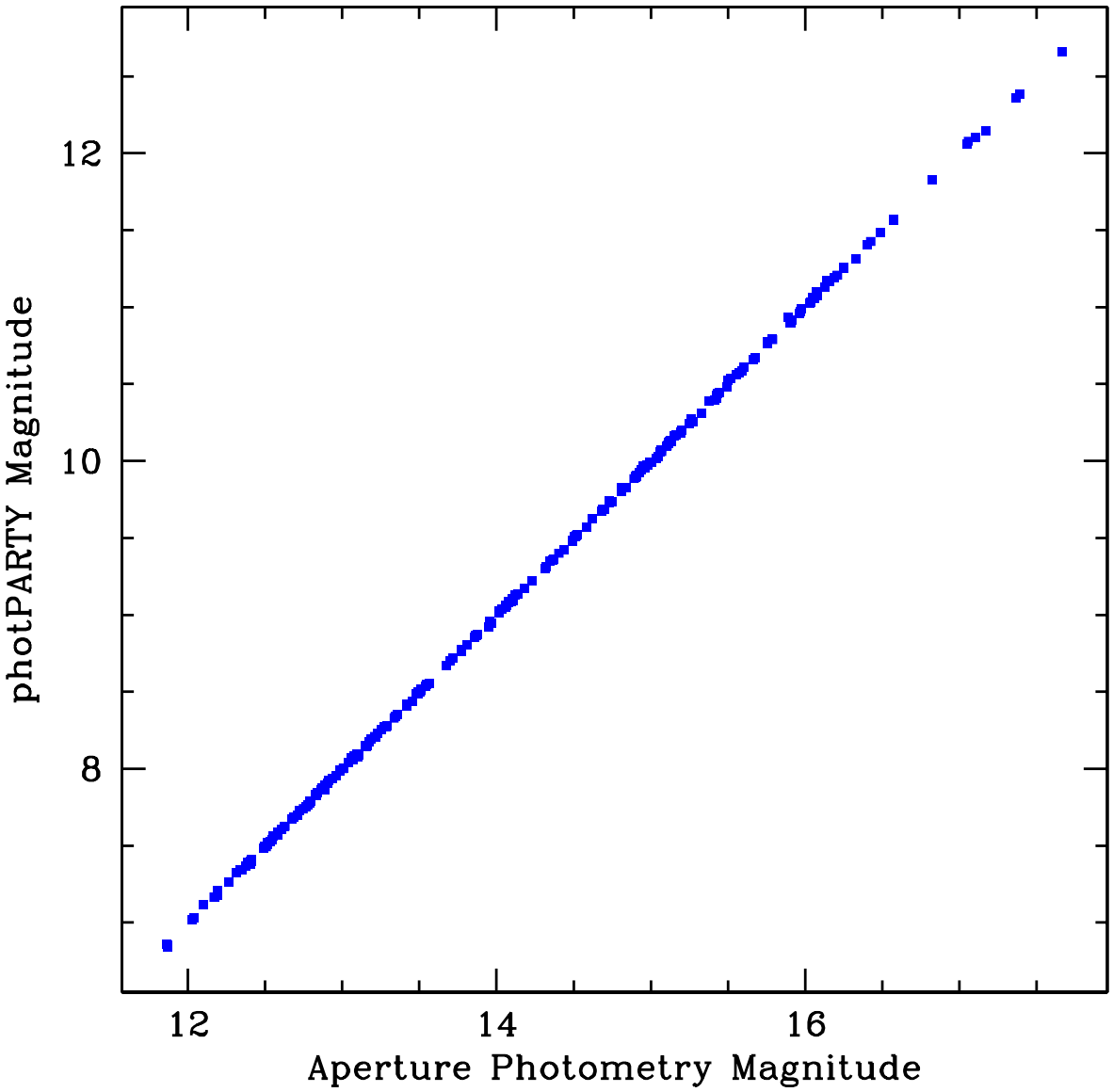

Figure 10: Comparison of photPARTY magnitudes to aperture photometry measurements from IRAF.

different measurements better than 0.01 magnitudes. Further spread is seen towards fainter magnitudes. This spread indicates that for the faintest stars, IRAF measurements are too bright or photPARTY measurements are too faint. A potential issue is the way bad pixels are rejected.

Where values above or below a certain threshold are replaced with zero. When these values are included in the square apertures used to compute magnitudes, any rejected pixels are still counted in the total number even though they are not contributing to the total flux. This could cause total fluxes that are too small and thus magnitudes that are too faint, and could explain the spread between IRAF and photPARTY for very faint magnitudes. Smaller stars with fainter magnitudes would be more affected by this issue due to having fewer pixels per star. A potential

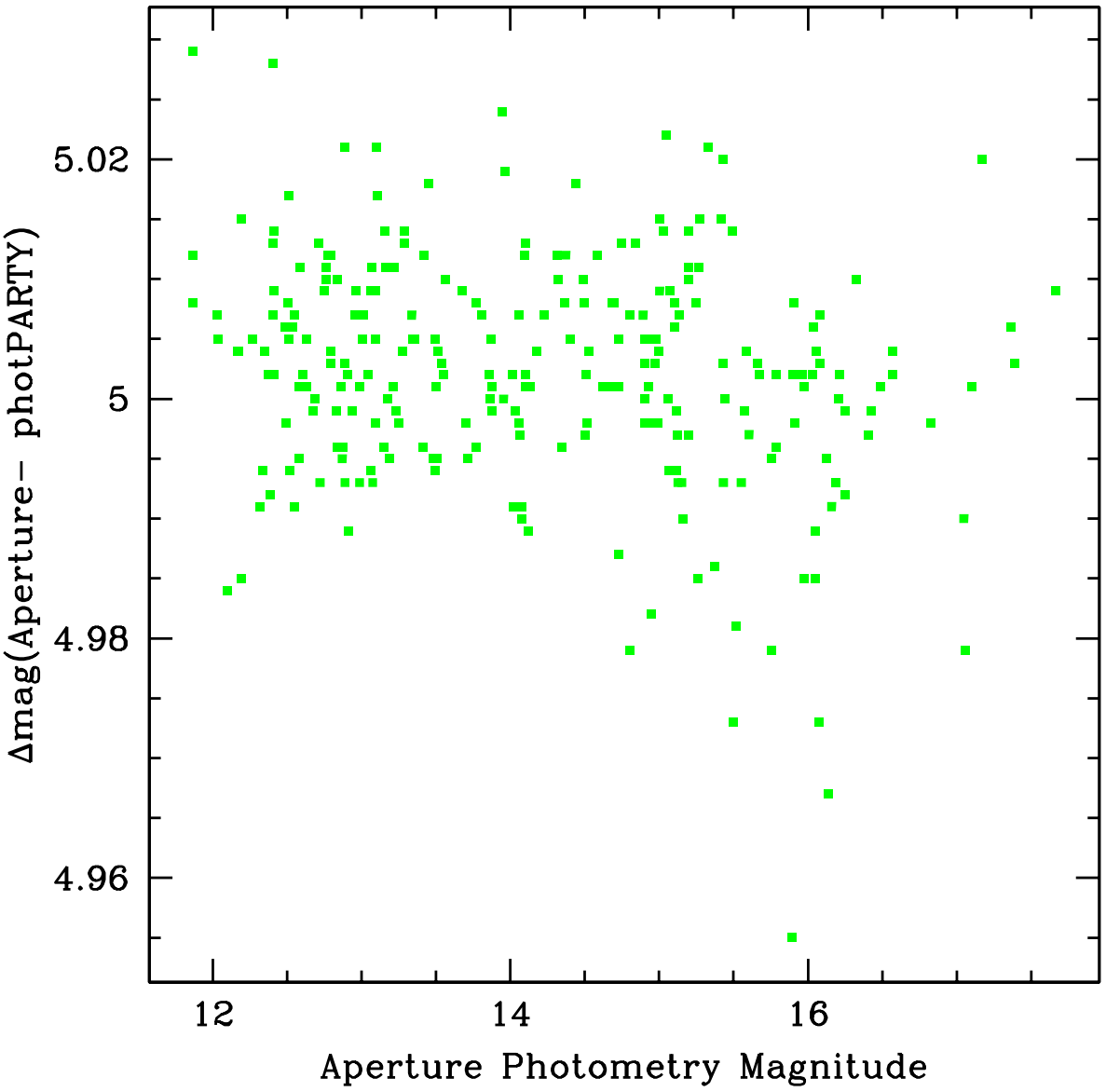

Figure 11: Spread between IRAF aperture photometry and photPARTY values for a range of magnitudes.

future improvement could track the number of rejected pixels and scale up the flux accordingly for more accurate faint magnitudes. Figure 10 shows a comparison to standard values for the stars in the set of frames. PhotPARTY magnitudes were corrected for extinction, which is the scattering and absorption of light caused by dust and gas, and compared to standard V-band magnitudes. Here again good agreement can be seen with a slight offset due to features of the

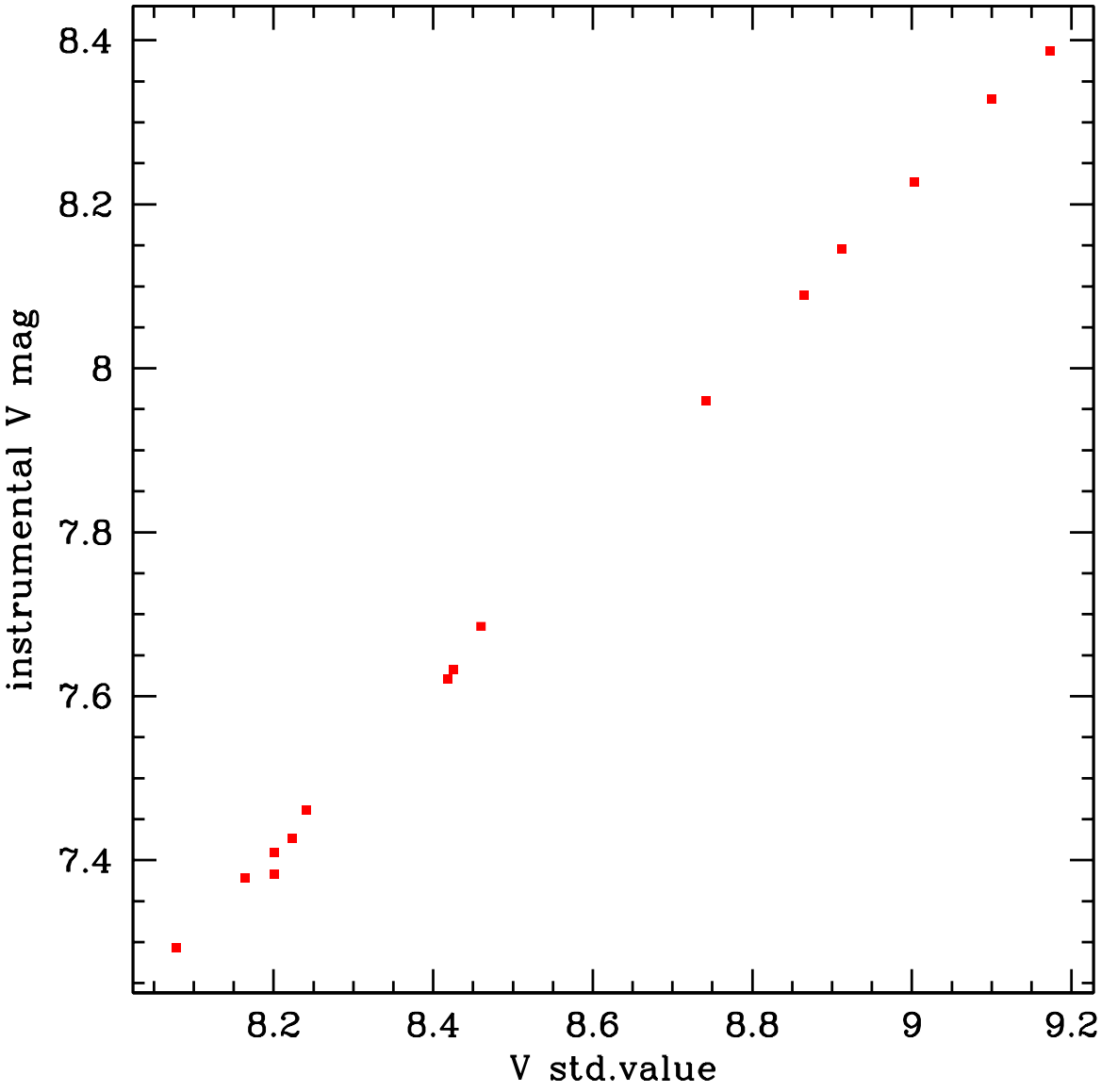

Figure 12: Standard values for V-band magnitudes compared to photPARTY extinction-corrected V-band magnitudes.

telescope and CCD used to take the data. The intercept of this line provides a means to transform all instrumental magnitudes from this data set into calibrated standard values. In Figure 11, a comparison is made between b-y color values for both photPARTY and standard values. Each color value is based on two separate magnitudes: one each in the 'b' and 'y' filter bands. This line has a slope of 0.9245 and a root mean square spread of 0.0066, indicating agreement even when two magnitudes in different filters are used to compare to standard values. For a set of 252

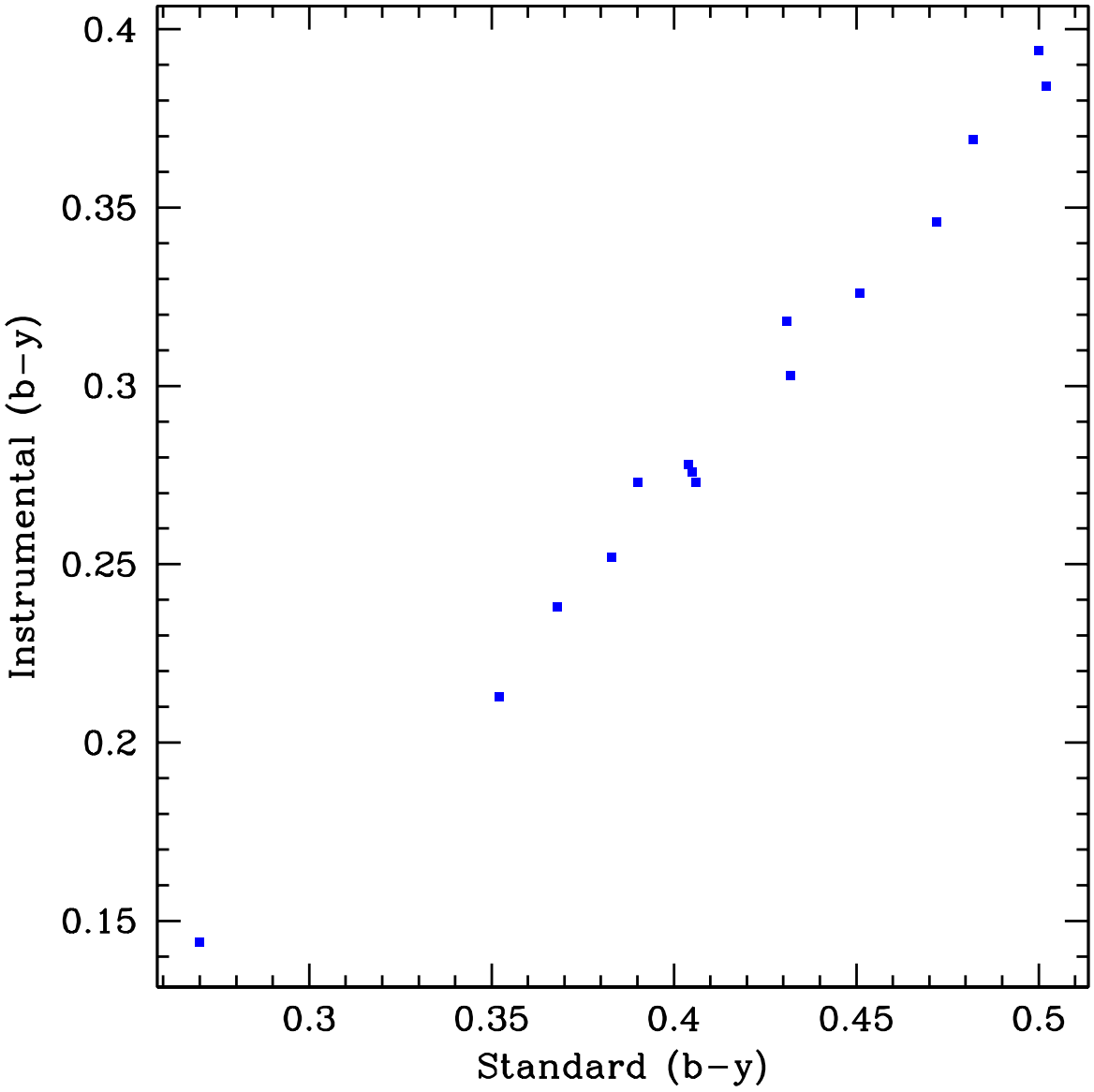

Figure 13: Color value comparison between photPARTY values and standard values.

stars, the standard deviation between measurements taken with photPARTY and those done in IRAF was 0.0223 magnitudes. For 36 common stars, the standard deviation between airmass-corrected photPARTY magnitudes and standard values was 0.0227 magnitudes. The small differences between IRAF and photPARTY magnitudes are negligible, while the potential for time saved is enormous. The manual IRAF photometry, performed by undergraduate student Kaitlin Neill, took upwards of 10 hours, while photPARTY's runtime was approximately four minutes, plus a small amount of time for initial calibration and determination of settings.

3.2 Online Availability

To make photPARTY available for interested users, the source files are maintained on GitHub at https://tasymons.github.io/photparty along with an explanation of requirements and directions to run the program.

\

References

Bradley, L., Sipocz, B., Robitaille, T., Tollerud, E., Vinícius, Z., Deil, C., Barbary, K., Günther, H. M., Cara, M., Droettboom, M., Bostroem, A., Bray, E., Bratholm, L. A., Pickering, T. E., Craig, M., Barentsen, G., Pascual, S., adonath, Greco, Kerzendorf, J. W., Littlefair, S., Ferreira, L., D'Eugenio, F., & Weaver, B. A. (2016). astropy/photutils: v0.3. doi:10.5281/zenodo.164986

Carroll, B. W. & Ostlie, D. A. (2007). An introduction to modern astrophysics. *Pearson Education, Inc.*, 2nd ed.

Chromey, F. R. (2016). To measure the sky: An introduction to observational astronomy. *Cambridge University Press*, 2nd ed.

Davis, L. E. (1994). A reference guide to the IRAF/DAOPHOT package.

Olsen, E. H. (1983). Four colour uvby Hβ photometry of A5 to G0 stars brighter than m=8.3. *A&AS*, 54, 55.

Olsen, E. H. (1993). Stromgren 4-colour uvby photometry of G5-type HD stars brighter than m_v=8.6. *A&AS*, 102, 89.

Olsen, E. H. (1994). Stromgren photometry of F- and G-type stars brighter than V=9.6. I. uvby photometry. *A&AS*, 106, 257.

Stromgren, B. (1966). Spectral classification through photoelectric narrow-band photometry. *ARA&A*, 4, 433.

Tody, D. (1986). The IRAF data reduction and analysis system. *Proc. SPIE Instrumentation in Astronomy VI*, 627, 733.

Tody, D. & Davis, L. E. (1992). CCDPHOT: an IRAF based real time CCD photometer. *ASP Conf. Ser.*, 25, 484.

The Astropy Collaboration, Robitaille, T. P., Tollerud, E. J., Greenfield, P., Droettboom, M., Bray, E., Aldcroft, T., Davis, M., Ginsburg, A., Price-Whelan, A. M., Kerzendorf, W. E., Conley, A., Crighton, N., Barbary, K., Muna, D., Ferguson, H., Grollier, F., Parikh, M. M., Nair, P. H., Günther, H. M., Deil, C., Woillez, J., Conseil, S., Kramer, R., Turner, J. E. H., Singer, L., Fox, R., Weaver, B. A., Zabalza, V., Edwards, Z. I., Bostroem, K. A., Burke, D. J., Casey, A. R., Crawford, S. M., Dencheva, N., Ely, J., Jenness, T., Labrie, K., Lim, P. L., Pierfederici, F., Pontzen, A., Ptak, A., Refsdal, B., Servillat, M., & Streicher, O. (2013). Astropy: A community Python package for astronomy. *A&A*, 558, A33.

Twarog, B. A. & Anthony-Twarog, B. J. (1995). Ca II H and K photometry on the uvby system. II. The catalog of observations. *AJ*, 109, 2828.

Wells, L. A. (1994). Photometry using IRAF.

Appendices

Appendix A: *photparty.py*

```
#The goal of this script is to read fits files, locate stars, and return magnitudes for those stars
#Background sky level is calculated using random median sampling of square arrays
#A specific inset of the data array is taken if desired
#Pixel values are summed row-wise and column-wise and background level is used to determine
which rows and columns contain stars
#A list of matched star coordinates is prepared
#Star magnitudes are obtained using square apertures after being background subtracted for the
region specific to the star
#By Teresa Symons 2016

#USER-DEFINED PARAMETERS AND SETTINGS:

#Define path to folder containing files to be run
#All fits or fit files will be automatically included
#Output files will also be placed into this folder
path = '/Users/Andromeda/PycharmProjects/files'

#Header key words/values:
#If no keyword exists, enter 'NONE' and define value instead
#Exposure time
exptimekword = 'EXP_TIME'
```

```
exptime = 5
#Filter
filterkword = 'FILTER'
filter = 'b'
#Airmass
airmasskword = 'AIRMASS'
airmass = 100
#Gain
gainkword = 'NONE'
gain = 1

#Parameters for background sampling:
#Width/length in pixels of box for random background sampling to determine background value
backsize = 5
#Number of random background samples to take
backnum = 1000

#Selection of area of each frame to analyze:
#If area of interest is in the central 50% of frame, select 'half'
#If area of interest is entire frame, select 'whole'
#If custom area of interest is desired, select 'custom' and define range of X and Y coordinates
with (1,1) as the bottom left corner of the frame
framearea = 'custom'
```

```
xlow = 900

xhigh = 1500

ylow = 1200

yhigh = 1750


#Pixel rejection:

#Select pixel value above which values will be zeroed out

uplim = 45000

#Select number of sigma below which negative pixel values will be zeroed out

#Example: lowsig = 3 means pixel values less than -3*inset standard deviation will become 0

lowsig = 3


#Detection level:

#Select number of standard deviations above background required for star detection

sig = 25


#Suppress or display plots of summed rows/columns with detection level marked (on or off)

plotdetect = 'on'


#Suppress or display plots of fits image with detected star apertures overlaid (on or off)

plotstars = 'on'


#Square-aperture size:
```

```
#Select the half-width of the box used for photometry
boxhw = 25

#END USER-DEFINED PARAMETERS

#Import math, plotting, extraneous functions, and fits file handling:
import numpy as np
import os
from astropy.io import fits
from background import background
from binsum import binsum
from starlocate import starlocate
from starmed import starmed
from starphot import starphot
from astropy.table import Table
import matplotlib.patches as patches
import matplotlib.pylab as plt

#Create list of files to run based on defined path, ignoring all files that are not fit or fits
files = [f for f in os.listdir(path) if any([f.endswith('fit'), f.endswith('fits')]) if not f.startswith('.')]

#Alternatively, run list of files in a specified text file
#In this case files will output to the location of the main script
```

```
#files = [line.rstrip() for line in open('files.txt')]


#Run for all files in list
for i in files:
    #Define names for output files based on names of original files
    (name, ext) = os.path.splitext(i)
    newname = path+'/'+name+'info.txt'
    datname = path+'/'+name+'dat.txt'
    #Open output files
    f = open(newname,'w')
    df = open(datname,'w')
    #Open fits file
    filepath = path+'/'+i
    image = fits.open(filepath)


    #Retrieve data, exposure time, airmass, and filter from fits file:
    Data = image[0].data
    if exptimekword == 'NONE':
        etime = exptime
    else:
        etime = image[0].header[exptimekword]
    if filterkword == 'NONE':
        filter = filter
```

```
    else:
        filter = image[0].header[filterkword]
    if airmasskword == 'NONE':
        airmass = airmass
    else:
        airmass = image[0].header[airmasskword]
    if gainkword == 'NONE':
        gain = gain
    else:
        gain = image[0].header[gainkword]

    #Compute background sky level through random median sampling:
    #Inputs: data array, nxn size of random subarray to use for sampling, and number of desired
sampling iterations
    back, skyvals = background(Data,backsize,backnum)

    #Create desired inset of total data array:
    if framearea == 'half':
        #Find midpoint of image data
        mid = len(Data)/2
        #Take an inset of data that is half of area
        inset = Data[round(mid/2):3*round(mid/2),round(mid/2):3*round(mid/2)]
        xlow = 0
```

```
        xhigh = 0
        ylow = 0
        yhigh = 0
    if framearea == 'whole':
        #Use entire data array as inset
        inset = Data
        mid = 0
        xlow = 0
        xhigh = 0
        ylow = 0
        yhigh = 0
    if framearea == 'custom':
        inset = Data[ylow-1:yhigh-1,xlow-1:xhigh-1]
        mid = 0

    #Blanket removal of bad pixels above 45000 and 3*standard deviation below 0:
    inset[inset>uplim] = 0
    std = np.std(inset)
    inset[inset<-lowsig*std] = 0

    #Calculate sky background for specific inset:
    #Inputs: inset data array, nxn size of random subarray used for sampling, number of desired
sampling iterations
```

```
    insetback, insetskyvals = background(inset,backsize,backnum)

    #Compute summed row and column values for desired array by number of bins:
    #Inputs: inset data array, number of bins desired
    rowsum, colsum = binsum(inset,1)

    #Locate values in summed row and column vectors that are greater than desired sigma level above background:
    #Inputs: Data array, background level variable, desired sigma detection level, summed row vector, summed column vector
    starrow, starcol, backsum, std, sigma = starlocate(inset,insetback,sig,rowsum,colsum)
    if starrow == []:
        print('Error: No stars found in '+name+' by row - check threshold.')
    if starcol == []:
        print('Error: No stars found in '+name+' by column - check threshold.')

    # Plot summed row and column values with detection level marked:
    if plotdetect == 'on':
        plt.plot(rowsum)
        plt.plot((0, len(rowsum)), (backsum + sigma * std, backsum + sigma * std))
        plt.title('Summed Rows' + '-' + name)
        plt.xlabel('Row Index in Data Inset')
        plt.ylabel('Summed Row Value')
```

```
        plt.show()
        plt.plot(colsum)
        plt.plot((0, len(colsum)), (backsum + sigma * std, backsum + sigma * std))
        plt.title('Summed Columns' + '-' + name)
        plt.xlabel('Column Index in Data Inset')
        plt.ylabel('Summed Column Value')
        plt.show()


    #Take indices of detected star pixels and divide into sublists by individual star:
    #Return sublists of star indices, number of stars, and median pixel of each star
    #Pair star center row with star center column and return total number of pairs found
    #Inputs: vectors containing indices of detected star pixels for row and column and inset data
array
    rowloc, colloc, numstarr, numstarc, rowmed, colmed, starpoints, adjstarpoints =
starmed(starrow,starcol,inset,mid,xlow,ylow)


    #Take list of star coordinates and find summed pixel values within a square aperture of desired
size:
    #Also find background values for each star and subtract them from star values
    #Convert background-subtracted star values into fluxes and then magnitudes
    #Inputs: half-width of nxn square aperture, inset data array, vector containing star coordinates,
exposure time
```

```
    boxsum, starback, backsub, flux, mags, hw, magerr = starphot(boxhw, inset, starpoints, etime,
gain, name)

    #Output data table to file:
    n = len(mags)
    tname = [i]*n
    tfilter = [filter]*n
    tairmass = [airmass]*n
    tetime = [etime]*n
    x = [x for [x,y] in adjstarpoints]
    y = [y for [x,y] in adjstarpoints]
    t = Table([tname, tfilter, tairmass, tetime, x, y, mags, magerr], names=('File_Name', 'Filter',
'Airmass', 'Exposure_Time', 'X', 'Y', 'Magnitude', 'Mag_Err'))
    t.write(df,format='ascii')

    #Plot fits image with square apertures for detected stars overlaid
    if plotstars == 'on':
        fig, ax = plt.subplots(1)
        ax.imshow(Data, cmap='Greys',vmin=0,vmax=10)
        for i in range(0,len(x)):
            rect = patches.Rectangle(((x[i]-1-hw),(y[i]-1-hw)), 2*hw, 2*hw, linewidth=1,
edgecolor='r', facecolor='none')
            ax.add_patch(rect)
```

```
        if framearea == 'custom':
            rect = patches.Rectangle((xlow-1,ylow-1), xhigh-xlow, yhigh-ylow, linewidth=1,
edgecolor='b', facecolor='none')
            ax.add_patch(rect)
        plt.title(name)
        plt.show()



    #Printing of values to check program function to file
    print('Exposure time:', file=f)
    print(etime, file=f)
    print('Filter:', file=f)
    print(filter, file=f)
    print('Airmass:', file=f)
    print(airmass, file=f)
    print('Sky background level:', file=f)
    print(back, file=f)
    print('Shape of inset array:', file=f)
    print(np.shape(inset), file=f)
    print('Inset background sky level:', file=f)
    print(insetback, file=f)
    print('Summed background value for one row/column:', file=f)
    print(backsum, file=f)
```

```
print('Standard deviation of inset:', file=f)
print(std, file=f)
print('Detection level in sigma:', file=f)
print(sigma, file=f)
print('Indices of detected stars:', file=f)
print(starrow, file=f)
print(starcol, file=f)
print('Indices of detected stars divided into sublist by star:', file=f)
print(rowloc, file=f)
print(colloc, file=f)
print('Number of stars found by row/column:', file=f)
print(numstarr, file=f)
print(numstarc, file=f)
print('Median pixel of each star by row/column:', file=f)
print(rowmed, file=f)
print(colmed, file=f)
print('Paired indices of star centers:', file=f)
print(starpoints, file=f)
print('Total number of stars found:', file=f)
print(len(starpoints), file=f)
print('Coordinates of stars (x,y):', file=f)
print(adjstarpoints, file=f)
print('Width/Height of boxes:', file=f)
```

```
print(hw * 2, file=f)
print('Pixel sums for boxes around each star:', file=f)
print(boxsum, file=f)
print('Background values for each star:', file=f)
print(starback, file=f)
print('Background subtracted star values:', file=f)
print(backsub, file=f)
print('Flux of stars:', file=f)
print(flux, file=f)
print('Magnitudes for each star:', file=f)
print(mags, file=f)
```

Appendix B: *fixindex.py*

```
#This function adjusts edges for a subarray when they fall outside the range of existing data
#Inputs include the length of the data array (assumed to be square), beginning row of subarray,
ending row of subarray,
#beginning column of subarray, and ending column of subarray
#By Teresa Symons 2016


def fixindex(length, ra, rb, ca, cb):
    test = [ra, rb, ca, cb]
    new = []


    #For each of the 4 indices, check if any fall outside the array
    #If they do, replace them with the edge of the array
    for i in test:
        if i < 0:
            newindex = 0
        elif i > length:
            newindex = length
        else:
            newindex = i
        new.append(newindex)
```

```
    #If these adjustments result in either dimension being only one row or column wide, adjust so
that the second index
    #from the edge is used as an inside boundary
    if new[0] == new[1]:
        if new[0] == 0:
            new[1] = 1
        elif new[0] == length:
            new[0] = length - 1
    if new[2] == new[3]:
        if new[2] == 0:
            new[3] = 1
        elif new[2] == length:
            new[2] = length - 1

    #Return new, adjusted indices within the range of the data array
    return new[0], new[1], new[2], new[3]
```

Appendix C: *background.py*

```
#Function to find background level of desired data file
#By Teresa Symons 2016

#Import math and array index adjustment
import numpy as np
from random import randint
from fixindex import fixindex
from scipy import stats

def background(input, length, n):

    #Subarray function returns random subarray of desired nxn size
    def subarray(inputarray, slength):
        a = randint(0,len(inputarray))
        b = randint(0,len(inputarray))
        array = inputarray[a:a+slength, b:b+slength]
        #If random subarray is outside the range of the data array, indices are adjusted
        if array.size == 0:
            #This function takes the proposed edges of a subarray and checks if they fall outside the range of the original array
            #If they do, edges are moved to the edge of the original array
```

```
        #Inputs: length of original array (assumed to be square), first row, last row, first column,
last column of subarray
        a, b, c, d = fixindex(len(inputarray), a, a+slength, b, b+slength)
        array = inputarray[a:b,c:d]
    return array


#Determine background sky level
skyvals = []


#Take many random subarrays of desired size and find the median value
for i in range(0,n):
    small = subarray(input,length)
    m = np.nanmedian(small)
    #m = stats.mode(small,axis=None)
    #m = float(m[0])
    skyvals.append(m)


#Median of all sky values is taken as background level
back = np.nanmedian(skyvals)
#back = stats.mode(skyvals,axis=None)
#back = float(back[0])


return back, skyvals
```

Appendix D: *binsum.py*

```
#This is a function that takes a desired data array and number of bins and bins the data by both
row and column,
#summing for each bin on each axis
#By Teresa Symons 2016


#Import math
import numpy as np


def binsum(array,bins):


    #For a desired bin number, bin data and sum row and column values in bins
    rowsum = []
    colsum = []
    binnum = int(round(len(array)/bins))
    for j in range(0,binnum):
        rowsum.append(np.sum(array[j*bins:j*bins+bins,:]))
        colsum.append(np.sum(array[:,j * bins:j * bins + bins]))


    return rowsum, colsum
```

Appendix E: *starlocate.py*

```
#This function takes a desired data array, computed background level, desired sigma detection level, and summed row
#and column vectors and locates indices in those vectors where the summed value is the desired sigma level above the
#computed summed background level
#By Teresa Symons 2016

def starlocate(array,background,sigma,summedrows,summedcols):
    #Import math
    import numpy as np

    #Calculate estimated background level summed over whole row/column
    backsum = background*len(array)

    starrow = []
    starcol = []

    #Calculate standard deviation of desired array
    std = np.std(array)

    #For both the summed row and column values, locate values greater than n sigma above background level
```

```
for k in range(0,len(summedrows)):
    if summedrows[k]>sigma*std+np.abs(backsum):
        starrow.append(summedrows.index(summedrows[k]))
    if summedcols[k]>sigma*std+np.abs(backsum):
        starcol.append(summedcols.index(summedcols[k]))


return starrow, starcol, backsum, std, sigma
```

Appendix F: *starmed.py*

```
#This function takes lists of row and column indices where all star exist and divides them into sublists by star
#It also returns the number of stars by row and by column, the median pixel values for each star, and paired coordinates for each star
#By Teresa Symons 2016


def starmed(starrow,starcol,inset,mid,xlow,ylow):
    #Import math
    import numpy as np


    #Divide all star location indices by star
    rowloc = []
    colloc = []
    starr = []
    starc = []
    starr.append(starrow[0])
    starc.append(starcol[0])
    #Start adding star indices to a list
    #If difference between previous and next index is greater than 1, assume new star begins and start new list of indices
    for i in range(1,len(starrow)):
        if starrow[i] == starrow[i-1] + 1:
```

```
        starr.append(starrow[i])
    elif starrow[i] == starrow[i-1] + 2:
        starr.append(starrow[i])
    else:
        rowloc.append(starr)
        starr = []
        starr.append(starrow[i])
rowloc.append(starr)

for j in range(1,len(starcol)):
    if starcol[j] == starcol[j-1] + 1:
        starc.append(starcol[j])
    elif starcol[j] == starcol[j-1] + 2:
        starc.append(starcol[j])
    else:
        colloc.append(starc)
        starc = []
        starc.append(starcol[j])
colloc.append(starc)

#Calculate number of stars (sublists) found by both row and column
numstarr = len(rowloc)
numstarc = len(colloc)
```

```
    #Find median pixel value of each star by row and column
    #If a star as only one pixel, include that as median value
    rowmed = []
    colmed = []
    for k in range(0,len(rowloc)):
        if len(rowloc[k]) == 1:
            rowmed.append(rowloc[k][0])
        else:
            rowmed.append(int(round(np.nanmedian(rowloc[k]))))
    for k in range(0,len(colloc)):
        if len(colloc[k]) == 1:
            colmed.append(colloc[k][0])
        else:
            colmed.append(int(round(np.nanmedian(colloc[k]))))



    #Check original data array for maximum column value associated with each star's row
coordinate
    #If column value appears within +/- one pixel in list of column coordinates, add coordinate
pair to list of star coordinates
    starpoints = []
    for i in rowmed:
```

```
j = np.argmax(inset[i, :])
if j in colmed:
    pt = []
    pt.append(i)
    pt.append(j)
    starpoints.append(pt)
elif j+1 in colmed:
    pt = []
    pt.append(i)
    pt.append(j)
    starpoints.append(pt)
elif j-1 in colmed:
    pt = []
    pt.append(i)
    pt.append(j)
    starpoints.append(pt)
elif j+2 in colmed:
    pt = []
    pt.append(i)
    pt.append(j)
    starpoints.append(pt)
elif j-2 in colmed:
    pt = []
```

```
        pt.append(i)
        pt.append(j)
        starpoints.append(pt)
    elif j+3 in colmed:
        pt = []
        pt.append(i)
        pt.append(j)
        starpoints.append(pt)
    elif j-3 in colmed:
        pt = []
        pt.append(i)
        pt.append(j)
        starpoints.append(pt)
    elif j+4 in colmed:
        pt = []
        pt.append(i)
        pt.append(j)
        starpoints.append(pt)
    elif j-4 in colmed:
        pt = []
        pt.append(i)
        pt.append(j)
        starpoints.append(pt)
```

```
    elif j+5 in colmed:
        pt = []
        pt.append(i)
        pt.append(j)
        starpoints.append(pt)
    elif j-5 in colmed:
        pt = []
        pt.append(i)
        pt.append(j)
        starpoints.append(pt)

#Adjust coordinates for original image and python indexing
adjstarpoints = [[xlow+y+round(mid/2)+1,ylow+x+round(mid/2)+1] for [x,y] in starpoints]

return rowloc, colloc, numstarr, numstarc, rowmed, colmed, starpoints, adjstarpoints
```

Appendix G: *starphot.py*

```
#This function takes a list of star coordinates and finds the sum of the pixel values for each star within a nxn box
#It also finds the background median for each star using four nxn boxes at the corners of each star's box
#The background value is subtracted from each star's value, converted into flux by dividing by exposure time,
# and this is then converted into a magnitude for each star
#By Teresa Symons 2016

def starphot(hw,inset,starpoints,etime,gain, name):
    #Import math and array index adjustment
    import numpy as np
    from fixindex import fixindex

    boxsum = []
    starback = []
    for i in starpoints:
        medsum = []
        #For each star, define a square box around it with the desired half-width
        box = inset[i[0]-hw:i[0]+hw,i[1]-hw:i[1]+hw]
        #If part of box extends outside of inset range, adjust size of box
        if box.size == 0:
```

```
        # This function takes the proposed edges of a subarray and checks if they fall outside the range of the original array
        # If they do, edges are moved to the edge of the original array
        # Inputs: length of original array (assumed to be square), first row, last row, first column, last column of subarray
        a, b, c, d = fixindex(len(inset), i[0]-hw,i[0]+hw,i[1]-hw,i[1]+hw)
        box = inset[a:b,c:d]


    #Define four boxes of the same size at each corner of the star box, making similar adjustments where necessary
    ul = inset[i[0]-3*hw:i[0]-hw,i[1]-3*hw:i[1]-hw]
    if ul.size == 0:
        a, b, c, d = fixindex(len(inset), i[0]-3*hw, i[0]-hw, i[1]-3*hw, i[1]-hw)
        ul = inset[a:b, c:d]
    ur = inset[i[0] -3*hw:i[0] -hw, i[1] +hw:i[1] +3*hw]
    if ur.size == 0:
        a, b, c, d = fixindex(len(inset), i[0] -3*hw, i[0] -hw, i[1] +hw, i[1] +3*hw)
        ur = inset[a:b, c:d]
    ll = inset[i[0] +hw:i[0] +3*hw, i[1] - 3 * hw:i[1] - hw]
    if ll.size == 0:
        a, b, c, d = fixindex(len(inset), i[0] +hw, i[0] +3*hw, i[1] - 3 * hw, i[1] - hw)
        ll = inset[a:b, c:d]
    lr = inset[i[0] +hw:i[0] +3*hw, i[1] +hw:i[1] +3* hw]
```

```
    if lr.size == 0:
        a, b, c, d = fixindex(len(inset), i[0] +hw, i[0] +3*hw, i[1] +hw, i[1] +3* hw)
        lr = inset[a:b, c:d]

    #Record medians of values inside boxes
    medboxul = np.nanmedian(ul)
    medboxur = np.nanmedian(ur)
    medboxll = np.nanmedian(ll)
    medboxlr = np.nanmedian(lr)

    #Sum up the values in the star box and take the median of all the background boxes
    medsum.append(medboxul)
    medsum.append(medboxur)
    medsum.append(medboxll)
    medsum.append(medboxlr)

    #Declare the background for a given star to be the median of the 4 median boxes
    starback.append(((2*hw)*(2*hw))*(np.nanmedian(medsum)))
    boxsum.append(np.sum(box))

#Subtract the background value from each star
backsub = [a-b for a, b in zip(boxsum,starback)]
```

```
    #Error message for stars with negative values after background subtraction
    if any(x<0 for x in backsub):
        print('Error: '+name+' contains a star with negative background-subtracted value - no
magnitude calculated.')

    #Calculate mag errors
    magerr = [1/np.sqrt(x) for x in backsub]
    #Convert to flux by dividing by exposure time
    flux = [x/etime for x in backsub]

    #Convert to magnitude
    mags = [-2.5*np.log10(a)+20 for a in flux]

    return boxsum, starback, backsub,flux, mags, hw, magerr
```